```cpp
//-----------------------------------------------------------------------------------------------------------------
// SEQUENTIAL CONTAINERS
//-----------------------------------------------------------------------------------------------------------------
#include <iostream>
#include <vector>
#include <deque>
#include <list>
#include <forward_list>

using std::vector;
using std::deque;
using std::list;
using std::forward_list;
using std::cout;
using std::endl;
void print(const std::vector<int> & v);
void print(const std::deque<int> & d);
void print(const std::list<int> & l);
void print(const std::forward_list<int> & fl);

int main(){
    std::vector<int> v;
    std::deque<int> d;
    std::list<int> l;
    std::forward_list<int> fl;

    // all containers are empty
    // let's store 1,2,3

    v.push_back(1);
    v.push_back(2);
    v.push_back(3);

    l.push_back(2);
    l.push_back(3);
    l.push_front(1); // it is efficient in a vector it is not possible  (no push_front)
                     // (it would be inefficient). Same considerations for forward_list

    fl.push_front(3); // fl doesn't have push_back, push_front in reverse order
    fl.push_front(2);
    fl.push_front(1);

    // for deque we can do same, let's go in reverse order to enjoy efficient push_front!
    d.push_front(3);
    d.push_front(2);
    d.push_front(1);

    // writing  elements to cout (read only)
    cout << "Print v, d, l, fl" << endl;

    print(v);
    print(d);
    print(l);
    print(fl);

    // let's change first element through reference fo first element
    vector<int>::reference rv1=v.front();
    rv1++;
    deque<int>::reference rd1=d.front();
    rd1++;
    list<int>::reference rl1=l.front();
    rl1++;
    forward_list<int>::reference rfl1=fl.front();
    rfl1++;

    // let's change last element through reference fo last element
    // we cannot do for forward_list!
    vector<int>::reference rv2=v.back();
    rv2++;
    deque<int>::reference rd2=d.back();
    rd2++;
    list<int>::reference rl2=l.back();
    rl2++;

    cout << "Print v, d, l, fl" << endl;
    print(v);
    print(d);
    print(l);
    print(fl);

    // let's print directly list first element
    cout << "Print l first element" << endl;
    list<int>::value_type i = l.front(); // i is a copy of the first element!
    cout << i << endl;

    // Copies
    vector<int> v2(v); //same as vector<int> v2 = v;
    deque<int> d2(v.cbegin(),v.cend());

    cout << "Print v2" << endl;
    print(v2);
    cout << "Print d2" << endl;
    print(d2);

    list<int> l2;
    l2.assign(l.cbegin(), l.cend());

    cout << "Print l2" << endl;
    print(l2);

    // Let's copy v to the end of v2
    v2.insert(v2.end(),v.cbegin(),v.cend());
    cout << "Print v2" << endl;
    print(v2);

    // Let's copy v at the beginning of d2
    d2.insert(d2.begin(),v.cbegin(),v.cend());
    cout << "Print d2" << endl;
    print(d2);

    // Let's copy 1, 2, 3 at the beginning of l2 and 5,6 at its end
    l2.insert(l2.begin(),{1,2,3});
    l2.insert(l2.end(),{5,6});
    cout << "Print l2" << endl;
    print(l2);

    // Let's resize d and v in a way they have 10 elements
    // (set elements to 30 in the second case)
    d.resize(10);
    v.resize(10,30);
    cout << "Print d" << endl;
    print(d);
    cout << "Print v" << endl;
    print(v);

    // delete all element from back
    while (!v.empty())
        v.pop_back();

    // delete all element from front
    while (!d.empty())
        d.pop_front();

    // delete all elements with iterator limits
    l.erase(l.begin(),l.end());

    // delete all elements through clear
    fl.clear();

    // All others are deleted through destructors ;)!!!
    return 0;
}

// if you want to change elements rely on iterator instead of const_iterator, but same loops!
void print(const std::vector<int> & v){
    for(vector<int>::const_iterator it = v.cbegin(); it != v.cend(); it++)
        cout << *it << " ";
    cout << endl;
}

// for deque, list and forward_list is same!!!
void print(const std::deque<int> & d){
    for(deque<int>::const_iterator it = d.cbegin(); it != d.cend(); it++)
        cout << *it << " ";
    cout << endl;
}

void print(const std::list<int> & l){
    for(list<int>::const_iterator it = l.cbegin(); it != l.cend(); it++)
        cout << *it << " ";
    cout << endl;
}

void print(const std::forward_list<int> & fl){
    for(forward_list<int>::const_iterator it = fl.cbegin(); it != fl.cend(); it++)
        cout << *it << " ";
    cout << endl;
}
```